A Study of Historical Flight Software Error Incidents to Influence Fault Tolerant Design

2023 Flight Software Workshop

Lorraine Prokop, Ph.D.

Technical Fellow for Software
NASA Engineering and Safety Center
lorraine.e.prokop@nasa.gov

Flight Software Error Visualization

Flight Computer *without* Software Errors
(Credit NASA)

Flight Computer *with* Software Errors
(Credit NASA)

Introduction

- Motivation
  - Very little literature exists characterizing software errors in real-time avionic systems
    - *How, where, and why is software* most likely to fail?
- Purpose
  - Raise awareness of how software fails through historical study
  - Recommend improvements to software fault tolerant design based on historical study
- Outline
  - Discuss Software Failures - Common Cause, Failure Classes, Mitigation strategies
  - Review NASA requirements for Software Fault Tolerance
  - Review Historical Software Failures
  - Analyze failures and provide statistics
    - Erroneous vs. fail-Silent
    - Reboot recoverability likelihood
    - Code Location
    - Missing or unknown code?

Software Common Cause Failure

- What is Software "Common Cause" or "Common Mode" Failure?
  - In many avionic architectures, hardware replication into multiple "strings" is done for hardware fault tolerance
    - Common Example: Three or four identical processors running "lock step" or synchronized vote on their outputs
  - However, the *same software load* often runs on these multiple processors
    - In this case, a *single* software failure normally would affect all strings in the same way at the same time
    - If only one processor is used, then *any* software failure could be considered "common mode" or "common cause"

Software Failure Classes

- Consider Two classes of software common cause:
  - Fail Silent – Computers all stop outputting, Ex: simultaneous crash
  - Erroneous output – Software does the wrong thing
    - Broader class of errors that can manifest in loss of control
  - Both should be considered when designing for fault tolerance
- Why differentiate?
  - It is much easier to determine of the software has stopped or crashed
    - Most systems implement a watchdog timer
  - How can you determine if the automation or software is doing something *wrong*?
    - Implement an independent monitoring algorithm – what if *that* is wrong (?)
  - Current space systems approach these manifestations in different ways
    - Many use humans in the loop to monitor and determine if the software is behaving as expected

Software Failure Categories

- Fail-Silent Cause Examples (*loss of output*)
  - Operating System Simultaneous Halt
    - Memory access violations, arithmetic errors, memory leaks, coding/logic errors
  - CPU Hog – Process Starvation
    - Infinite loop, priority inversion, extreme latencies
- Erroneous Output Causes Examples (*wrong output*)
  - Coding/Logic Error
    - Missing or Wrong Requirements (many errors can be discussed as “missing” requirements)
    - Insufficient modeling of real-world
    - Latencies/Timing Errors, Task Overruns, real-time behavior misunderstood, not handled
  - Data Parameter Misconfigured
    - Wrong data input, database, Units, precision, sign
  - Unanticipated / Erroneous Sensor Input
  - Erroneous Command Input - Operator / Procedural Error

Software Common Cause Risk Mitigation Strategies

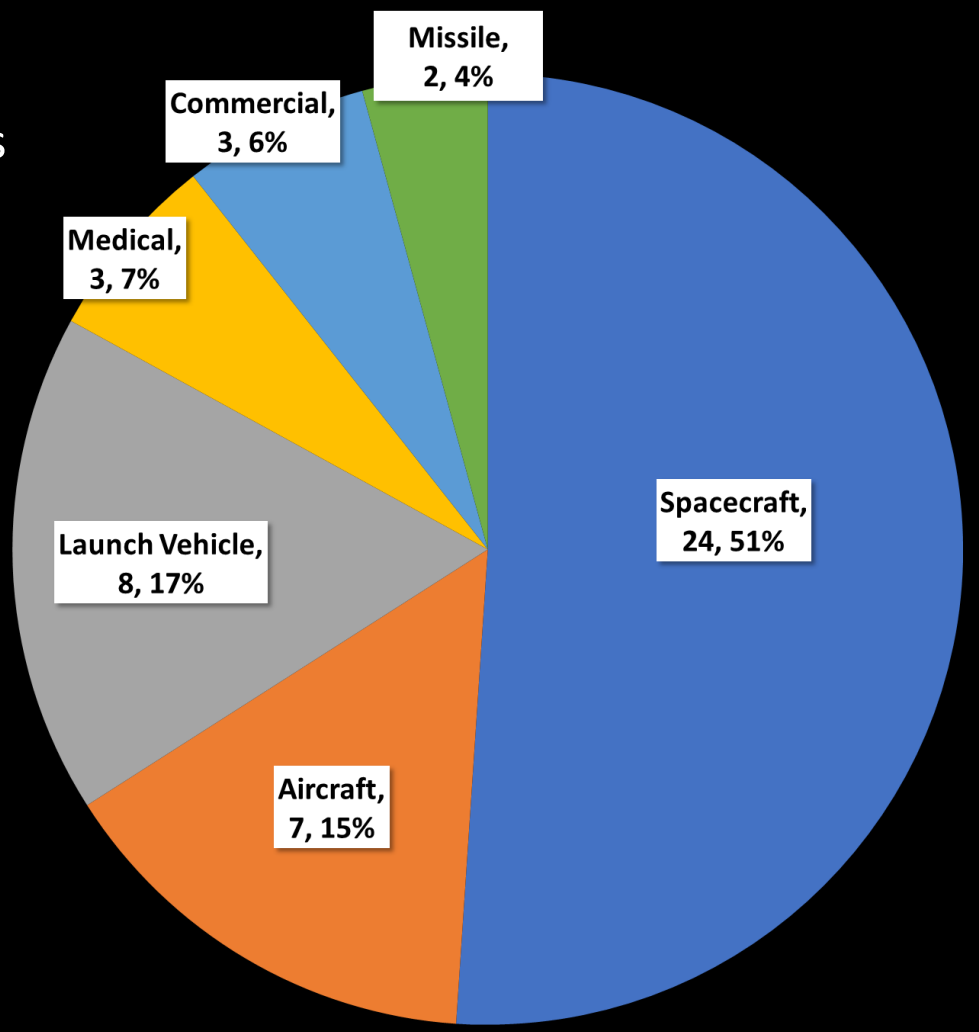
- Dynamic phases are riskiest times for software common cause due to short time-to-criticality
  - Ascent, Rendezvous, Entry
  - Mitigation strategies should be considered in relation to time-to-criticality, time-to-effect, time delay
  - During quiescent times with greater time-to-effect, there may be additional mitigation strategies available (i.e. software upload, reboot)
- Common mitigations for software common cause failures
  - Backup software – Set of dissimilar software automatically or manually engaged
    - Backup software systems can reside in same or different processor, depending on the failure cause
  - Manual piloting or control, crew/ground intervention
  - Reduced-capability primary software, software subset or “safe mode”
  - Uplink of new software
  - System Reboot

NASA Requirements for Software Fault Tolerance

- NPR 8705.2C: HUMAN-RATING REQUIREMENTS FOR SPACE SYSTEMS
  - 3.2.3 The space system shall provide at least single failure tolerance to catastrophic events, with specific levels of failure tolerance and implementation (similar or dissimilar redundancy) derived via an integration of the design and safety analysis.
  - 3.2.4 The space system shall provide the failure tolerance capability in 3.2.3 without the use of emergency equipment and systems.
  - 3.2.7 The space system shall provide the capability to mitigate the hazardous behavior of critical software where the hazardous behavior would result in a catastrophic event.
  - 3.3.2 The crewed space system shall provide the capability for the crew to manually override higher level software control and automation (such as automated abort initiation, configuration change, and mode change) when the transition to manual control of the system will not cause a catastrophic event.
- NPR 7150.2D: NASA SOFTWARE ENGINEERING REQUIREMENTS
  - 3.7.3 If a project has safety-critical software or mission-critical software, the project manager shall implement the following items in the software: [SWE-134] …
    - No single software event or action is allowed to initiate an identified hazard. …
- Software Assurance Standards to Assure these Requirements are Met:
  - NPR 8739.8A: SOFTWARE ASSURANCE AND SOFTWARE SAFETY STANDARD
  - NASA-STD-8719.13B: SOFTWARE SAFETY STANDARD

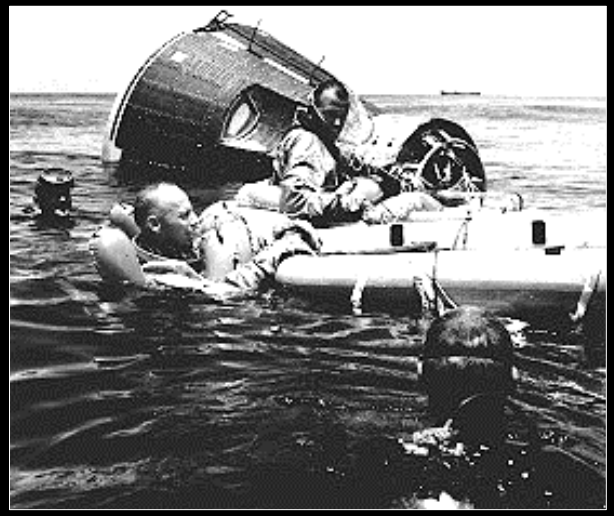
Historical Analysis

- Historic Failure Incidents Involving Software
  - We studied software failure incidents primarily at NASA and aerospace – when automation did not behave as expected
    - 47 incidents were characterized – since beginning of computers
      - Aerospace (41) – loss of life, mission, close-call
      - Non-Aerospace (6) –
        - 3 Medical (loss of life) , 3 Commercial (3) (loss of service)
    - We categorized software errors to determine:
      - Which is more prevalent – fail silent or erroneous?
      - Could the failure have been corrected by reboot?
      - Was this an unanticipated situation – missing code, wrong code, or unknown unknown?
      - *Where* in the code was the failure introduced?
  - **NOTE**: The *root cause* of these failures may not all be attributable to software (**why** it was programmed like that), but how the incident *initially manifested* during operations (**how** it behaved) is characterized



Historical Software Incidents (1962-1981)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
1962	**Mariner 1 Mission – Atlas-Agena**	Programmer error in ground guidance veered launch vehicle off course	Loss of vehicle	Erroneous Output	No	No	Code/Logic	No
1965	**Gemini 3**	Incorrect lift estimate causes short landing	Landed 84 km short, crew manually compensated, decreasing short landing error	Erroneous Output	No	Yes	Code/Logic	Yes
1965	**Gemini 5**	Data error of earth rotation lands Gemini 5 short	Landed 130 km short	Erroneous Output	No	No	Data	No
1968	**Apollo 8**	Memory Inadvertently Erased	Close Call fixed manually	Erroneous Output	No	No	Command Input	No
1969	**Apollo 10**	Switch Misconfigured as bad input data to abort guidance	Vehicle tumbled, close call, recovered manually	Erroneous Output	No	No	Data	No
1981	**STS-1**	Failure of computers to sync	Launch Scrub of First Shuttle flight	Fail Silent	Yes	Yes	Code/Logic	No



(Photo Credits: NASA)

Historical Software Incidents (1982-1994)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Recoverable?	Missing Code?	Error Location	Unknown-unknown?
1982	**Viking-1**	Erroneous Command caused loss of comm	End of mission	Erroneous Output	No	No	Command Input	No
1985-87	**Therac-25**	Radiation Therapy machine output lethal doses, user input speed	Four deaths, two chronic injured	Erroneous Output	No	No	Code/Logic	No
1988	**Phobos-1**	Erroneous unchecked uplinked command lost vehicle	Loss of vehicle/Mission	Erroneous Output	No	No	Command Input	No
1988	**Soyuz TM-5**	Wrong code executed to perform de-orbit burn	Extra day in orbit, New code uplinked	Erroneous Output	No	No	Code/Logic	No
1991	**Aries - Red Tigress I**	Bad command causes guidance error	Loss of Vehicle	Erroneous Output	No	No	Sensor Input	No
1991	**Patriot Missile**	Patriot failed target intercept due to 24-bit rounding error growth in time over time	Failed to intercept scud missile, resulting in American barracks being struck, 28 soldiers killed, 100 injured	Erroneous Output	Yes	No	Code/Logic	No
1992	**F-22 Raptor**	Software failed to compensate for pilot-induced oscillation in presence of lag	Loss of test vehicle	Erroneous Output	No	Yes	Sensor Input	Yes
1994	**Clementine Lunar Mission**	Erroneous thruster firing exhausted propellant, cancelling asteroid flyby	Failed mission objective	Erroneous Output	No	No	Code/Logic	No

Photo Credits: The National Archives, NAID: 6361754 (top), NAID: 6424495 (bottom)

Historical Software Incidents (1994-1999)

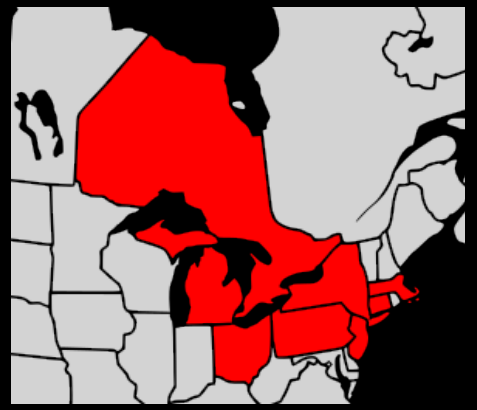
Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown - unknown
1994	**Pegasus XL STEP-1**	Booster loss of control due to lateral instability	Loss of vehicle/Mission	Erroneous Output	No	Yes	Code/Logic	Yes
1994	**Pegasus HAPS**	Navigation software error prematurely shut down upper stage	Unintended/low orbit	Erroneous Output	No	Yes	Code/Logic	No
1996	**Ariane 5 Maiden Flight**	Unprotected overflow in floating-point to integer conversion disrupted inertial navigation system	Loss of Vehicle	Erroneous Output	No	No	Code/Logic	No
1997	**Pathfinder**	Software priority inversion caused images to stall	Close Call for Mission Loss	Erroneous Output	No	No	Code/Logic	No
1998	**Delta III**	Unanticipated 4Hz Oscillation in control system led to vehicle loss	Loss of vehicle	Erroneous Output	No	Yes	Code/Logic	Yes
1999	**Mars Polar Lander**	Premature shut down of landing engine due to misinterpretation of landing signature	Loss of Vehicle/mission	Erroneous Output	No	Yes	Sensor Input	No
1999	**Mars Climate Orbiter**	Metric vs. imperial units error	Loss of vehicle/mission	Erroneous Output	No	No	Data	No

Mars Polar Lander (Credit: NASA)

Historical Software Incidents (1999-2003)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
1999	**Titan IV B Centaur**	Programming error omitting decimal in data file caused loss of control	Unintended orbit, Milstar Satellite lost 10 days after launch	Erroneous Output	No	No	Data	No
2000	**Zenit 3SL**	Ground software error failed to close valve.	Loss of Vehicle	Erroneous Output	No	No	Code/Logic	No
2001	**Pegasus XL/HyperX Launch Vehicle / X-43A**	Airframe failure due to inaccurate analytical models	Loss of vehicle/mission	Erroneous Output	No	Yes	Code/Logic	Yes
2001	**STS-108 through 110**	Shuttle main engine controller mix-ratio software coefficient sign-flip error	Significant close call, SME underperformance, though not extreme enough to not reach orbit.	Erroneous Output	No	No	Data	No
2003	**Multidata Systems Radiation Machine**	Radiation Therapy machine output lethal doses, counterclockwise user input	Many injured, 15 people dead.	Erroneous Output	No	No	Code/Logic	No
2003	**Soyuz - TMA-1**	Undefined yaw value triggered Ballistic reentry	landed 400 km short	Erroneous Output	No	No	Code/Logic	No
2003	**North American Electric Power Grid**	Real-time software errors contribute to Widespread power outage	Widespread Loss of Power Service (2 hr - 4 days)	Fail Silent	No	No	Code/Logic	No

STS-108 Crew (Credit: NASA)



Historical Software Incidents (2005-2008)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2005	**CryoSat-1**	Missing command causes loss of vehicle	Loss of Vehicle	Erroneous Output	No	Yes	Code/Logic	No
2005	**DART (Demonstration of Autonomous Rendezvous Technology)**	Navigation software errors fail mission objectives.	Loss of mission objectives	Erroneous Output	No	No	Code/Logic	No
2006	**Mars Global Surveyor (MGS)**	Erroneous command led to pointing error and power/vehicle loss	Premature Loss of vehicle	Erroneous Output	No	No	Code/Logic	No
2007	**F22 First Deployment**	International Date Line crossing crashed computer systems	Loss of navigation & communication	Fail Silent	No	Yes	Code/Logic	No
2008	**STS-124**	All 4 shuttle computers fail / disagree during fueling	Fueling stopped	Erroneous Output	No	Yes	Sensor Input	No
2008	**Quantas Flight 72, Airbus A330-303**	Sensor Input spikes caused autopilot to pitch-down, resulting in crew and passenger injuries	One crew member and 11 passengers suffered serious injuries	Erroneous Output	No	Yes	Sensor Input	Yes

Quantas Flight 72 (Credit: Masakatsu Ukon, CC BY-SA 2.0, via Wikimedia Commons).

Historical Software Incidents (2008-2017)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2008	**B-2 Spirit - Guam crash**	Miscalculation in flight computers with missing input data calculated uncommanded pitch up	Crew members successfully ejected.	Erroneous Output	No	Yes	Sensor Input	Yes
2012	**Red Wings Flight 9268 TU-204 crash**	Unanticipated landing circumstances coupled with design features resulted in crash landing	5 of 8 crewmembers killed	Erroneous Output	No	Yes	Code/Logic	Yes
2015	**Airbus A400M test flight**	Missing software parameters during installation cause crash	Four fatalities	Erroneous Output	No	No	Data	No
2015	**SpaceX CRS-7**	"Open Chute" command invalidated after launch vehicle failure	Possibly could have saved Dragon capsule from crash landing.	Erroneous Output	No	Yes	Code/Logic	No
2016	**Hitomi X-ray space telescope**	Error in computing spacecraft orientation led to spacecraft loss	Lost of vehicle	Erroneous Output	No	No	Code/Logic	No
2017	**SpaceX CRS-10**	Erroneous relative state vector transmitted to Dragon	ISS rendezvous delay	Erroneous Output	No	No	Data	No

CRS-7 Mishap (Credit: NASA)

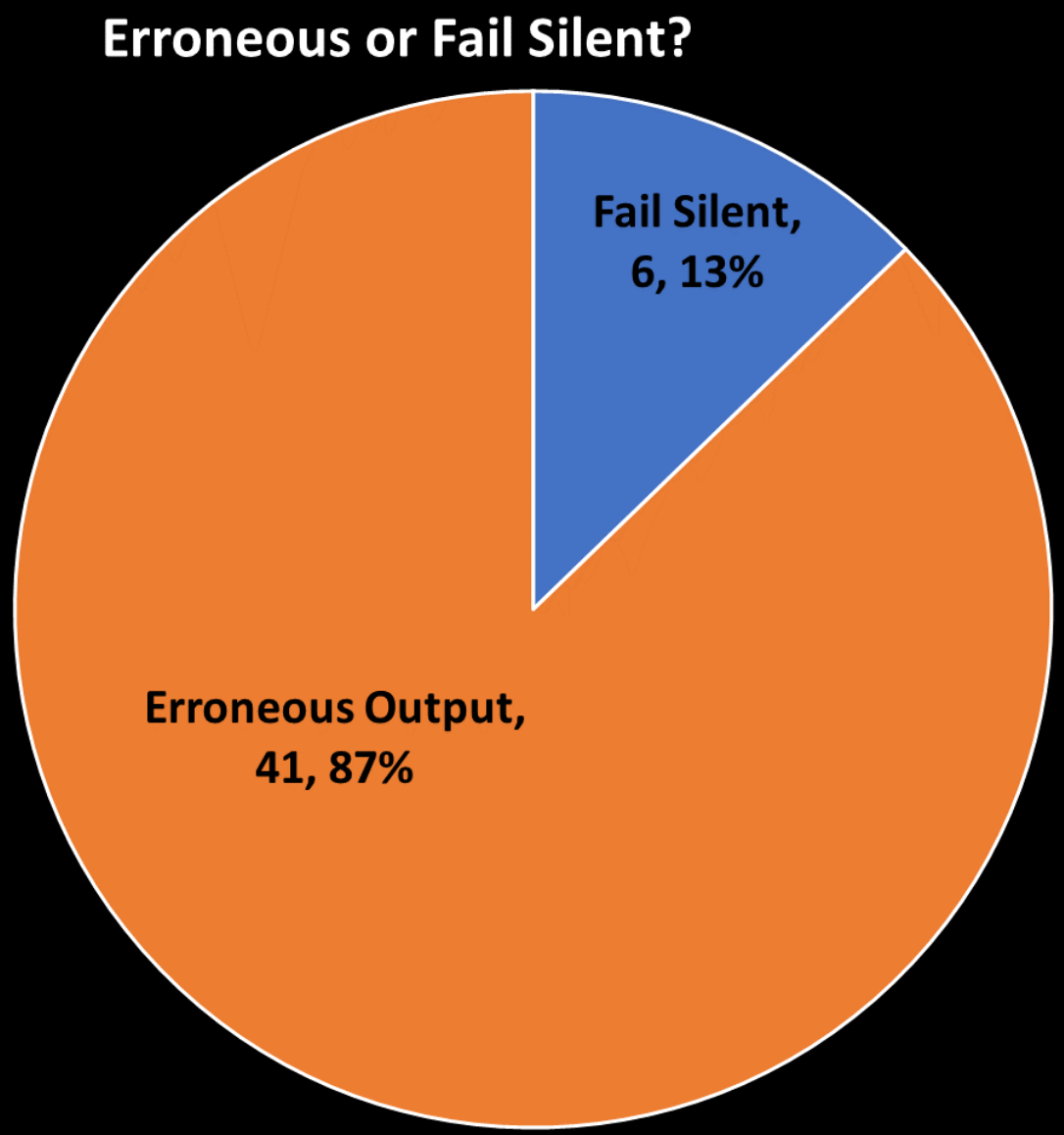
Historical Software Incidents (2018-2021)

Year	Flight or System	Title	Result / Outcome	Erroneous or Silent?	Reboot Fix?	Missing Code?	Error Location	Unknown-unknown
2018, 2019	**737 Max crash**	Unanticipated software response to faulty sensor input	346 people died on two flights	Erroneous Output	No	Yes	Sensor Input	Yes
2019	**Boeing Orbital Flight Test (OFT)**	Incorrect MET causes no ISS rendezvous and short mission, and uncovers other latent LOM software errors.	Failed ISS rendezvous, multi-year program delay	Erroneous Output	No	No	Code/Logic	No
2019	**Beresheet**	Reboots cause engine shutdown on lunar descent	Loss of vehicle	Fail Silent	No	No	Code/Logic	No
2020	**Amazon Web Service (AWS) Kinesis**	Maximum threads reached caused cascading server outage	Loss of service, revenues.	Fail Silent	No	Yes	Code/Logic	No
2020	**BD Alaris™ Infusion Pump**	Infusion delivery system software causes injury/death	55 injuries, 1 death	Erroneous Output	No	No	Code/Logic	No
2021	**Global Facebook Outage**	Bad command causes global Facebook and cascading communication outages.	Disrupted communication, loss of revenues	Fail Silent	No	No	Command Input	No
2021	**ISS**	Uncontrolled ISS attitude spin from erroneous thruster firing software	Close Call	Erroneous Output	No	No	Code/Logic	No

Boeing OFT Landing (Photo Credit: NASA)

And the List continues… 2023 NOTAM

Incident Statistics – Erroneous vs. Fail Silent



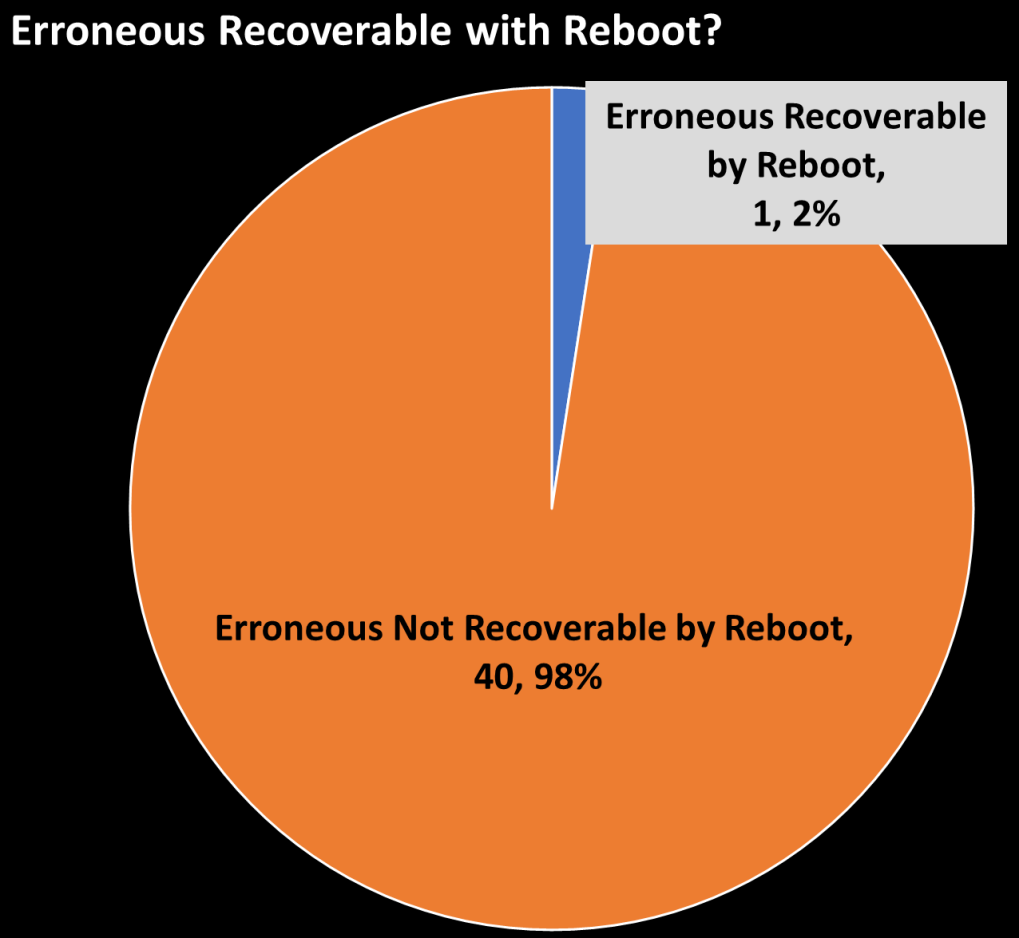
Takeaway:

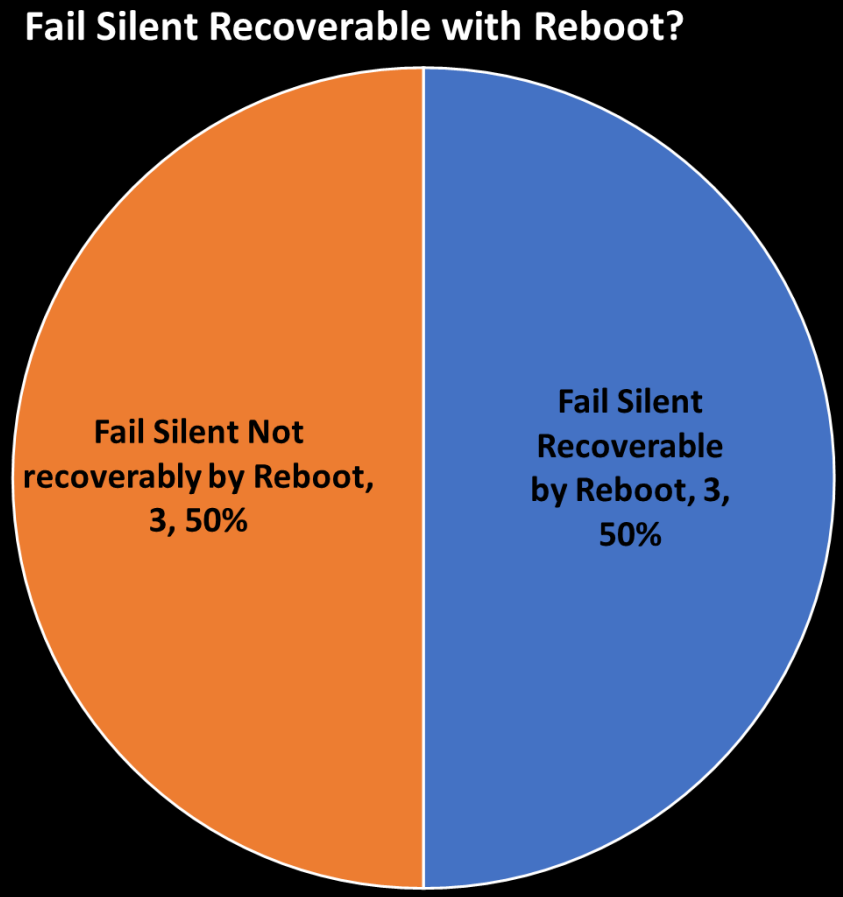
- ***Historically, erroneous output situations were much more prevalent than fail-silent cases***
- ***Roughly 90-10% rule of thumb***

Fault-tolerant Design Tip:

- ***Design should consider relative likelihoods of these manifestations***
- ***Systems should consider the question, "What is the risk of the software doing something wrong?" at critical moments***

Reboot Recoverability Likelihood
Erroneous vs. Fail Silent





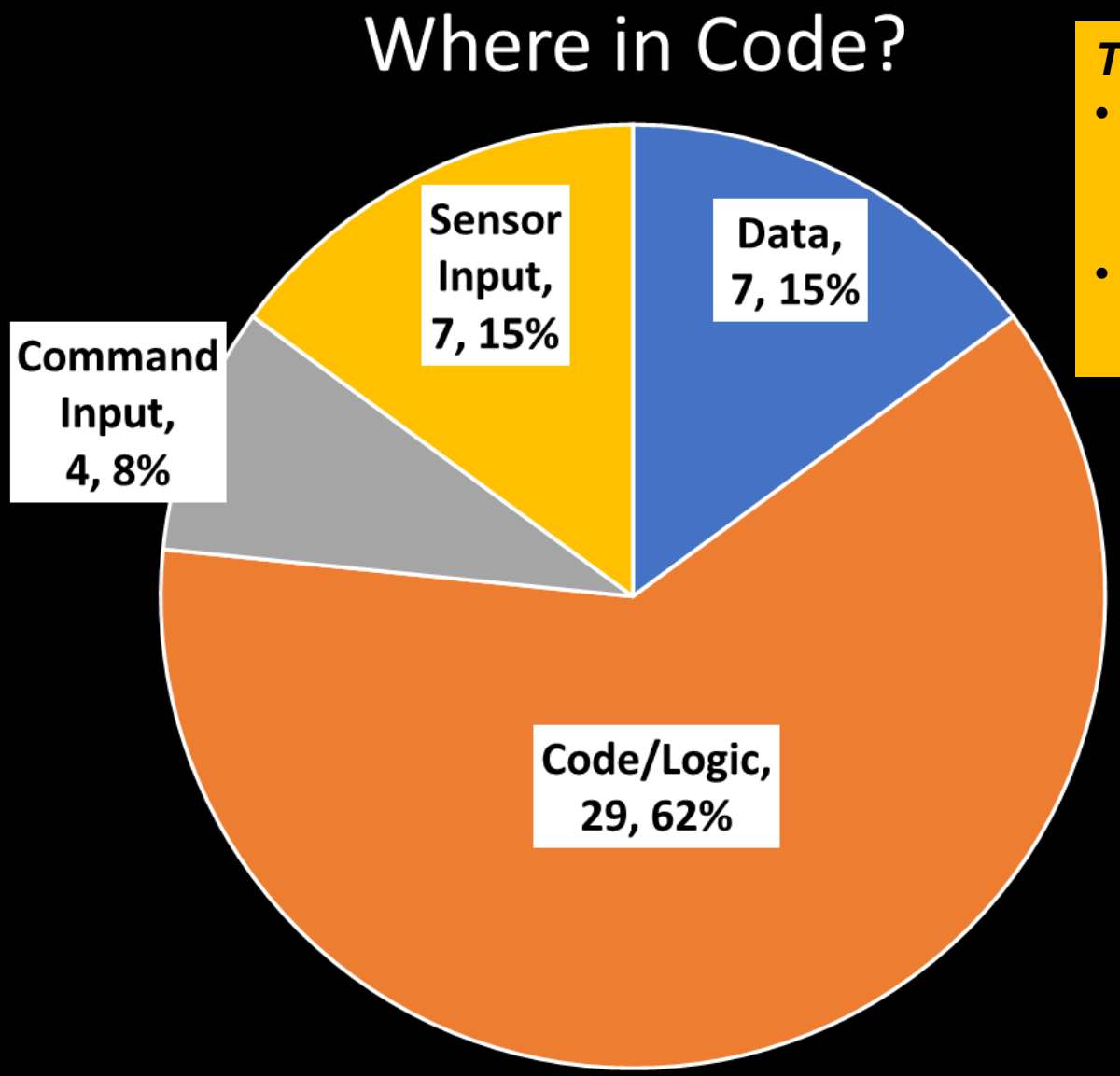
Takeaways:

- ***Rebooting is predominantly ineffective to clear/recover from erroneous output situations***
- ***Rebooting is a partial solution to clear fail-silent errors***

Fault-tolerant Design Tip:

- ***Do not rely on reboot to clear all software faults***

Incident Statistics – Software Architecture Error Location



Takeaway(s):

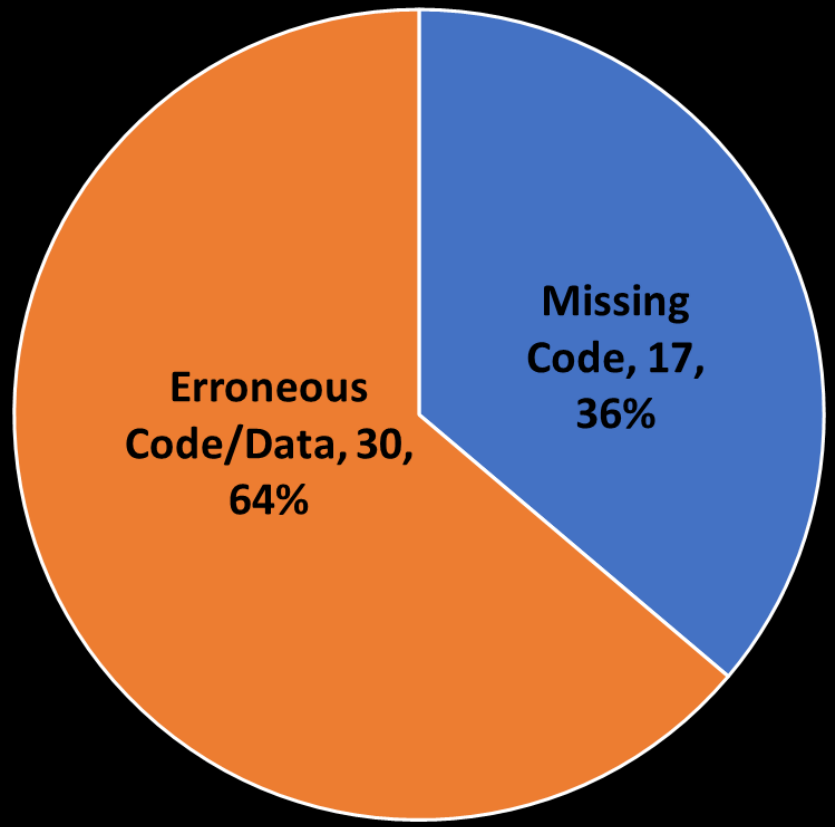
- ***Coding/logic (includes missing requirements and unknowns) and data configuration errors account for most software incidents***
- ***Input Errors – Command or Sensor Input Accounted for 23% of errors***

Fault-tolerant Code Tips:

- ***Project should test according to likelihoods***

- ***Code/Logic – off-nominal testing, peer review, unit testing, simulation/modeling***
- ***Data Misconfiguration – data validation prior to use, system expert review***
- ***Input Errors – Off-nominal or random input test generation***
  - ***Sensor input –hardware-in-the-loop testing***
  - ***Command input – validation, processes/procedures***

Incident Statistics – Absence of Code



Missing Code may arise from:

- Missing requirements
- Unanticipated situation
- Insufficient understanding or modeling of real world

- Adding code could have corrected the "missing code" incidents – *in hindsight*

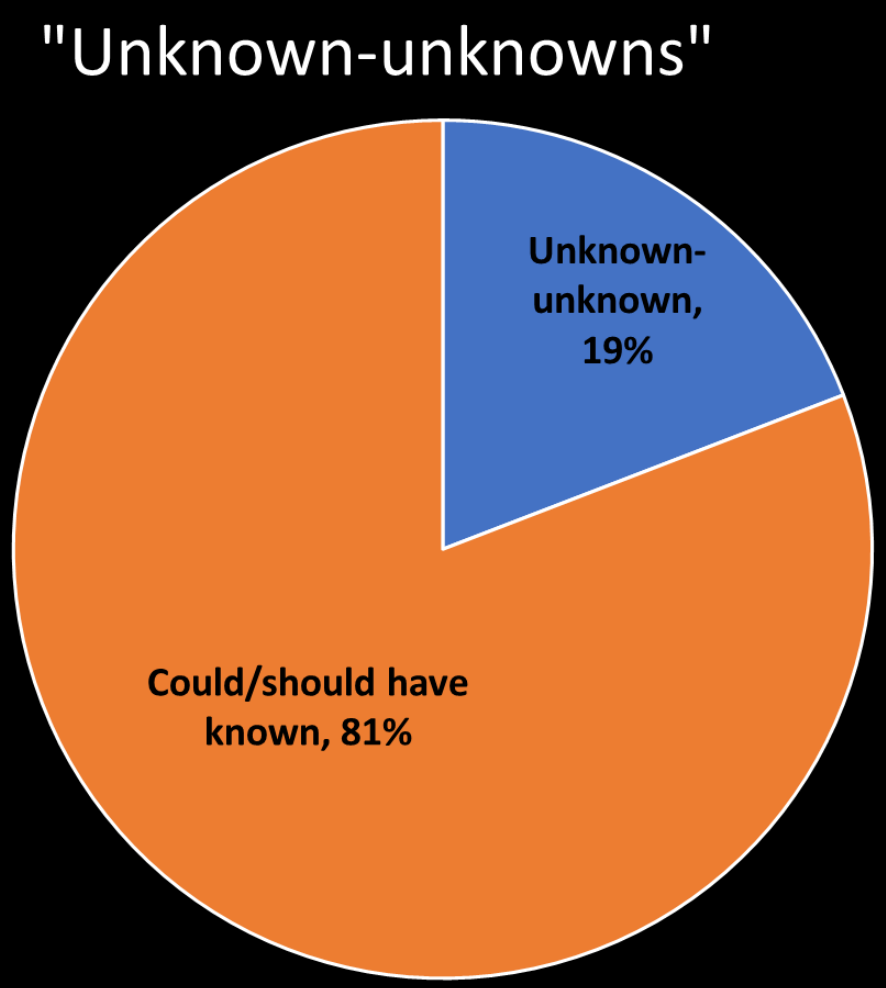
Fault-tolerant Design Tip:

- ***Projects should reserve test time to create off-nominal or unexpected conditions***

Takeaways:

- ***Even fully tested code does not uncover errors that arise from missing code/unanticipated situations***
- ***Hard to test code that is not there(!)***

Incident Statistics – “Unknown Unknowns”



Takeaways:

- ***Categorizing “unknown-unknowns” is highly subjective***
- ***Included here:***
  - ***unknown aero/handling, physics***
  - ***Insufficient modeling***
  - ***highly unusual input***
  - ***undetermined behavior in the presence of faults or multiple failures***

Fault-tolerant Design Tip:

- ***Backup systems can be considered to protect for “unknown- unknowns”***
- ***Projects should actively work to balance risk between “knowing everything” and project constraints (budget/schedule)***

Software Common Cause Failure Summary

- Software “common cause” or “common mode” errors occur when a single software error results in unexpected behavior, even if running on multiple strings
- Software in Space Systems should be architected for redundancy based on criticality and time-to-effect, with requirements driven primarily by NPR 8705.2C and NPR 7150.2D
- Software Errors manifest in two ways: Silent or Erroneous
- Study of historical software incidents indicates the following
  - Erroneous output is much more prevalent – roughly 90% of the incidents
  - Rebooting is largely ineffective to recover from erroneous situations, and partially effective for silent software
  - Software logic errors are the most common form, then data config, and 23% of errors arise from input
  - Missing Code accounted for 36% (including requirements, unknowns) of historic software errors
  - “Unknown-unknowns” account for roughly 20% of software error incidents, subjectively
- Fault-tolerant systems should be designed with these statistics in mind
  - Consider the Erroneous Case more than the fail silent
  - Don’t always rely on reboot
  - Employ hardware-in-the-loop, test-like-you-fly, and off-nominal testing
  - Validate configuration and command data prior to use
  - Consider use of backup strategies for critical events

References & Follow-on Work

- Prokop, Lorraine, E., “Software Error Incident Categorizations in Aerospace” [Manuscript submitted for publication]. NASA Engineering and Safety Center. 2023.

- Follow-on work:
  - This dataset can be used for further study, for example, to answer the following
    - What was the root cause of this error? (Why was the software programmed the way it was?)
    - Would a backup system have helped?
      - What kind of a backup system could have helped?”
      - Would a human-in-the-loop, a dissimilar backup, a monitor system, or no backup at all be best?
    - Was this a multi-string common-cause failure?
    - Was a manual or automated backup system used?,
    - What phase of the project could/should this incident been averted?
    - How much and what type of testing may have averted these errors?